Licenciatura Engenharia Informática e Multimédia

Modelação e Programação – MP

Relatório Trabalho Prático 3

Docente André Gomes

Trabalho realizado por:

Fábio Dias, nº 42921

Índice

1.xmlreadwrite

1.1. XmlReadWrite

Esta classe, disponibilizada pelo docente, apresenta um método main que tanto lê como escreve num documento XML.

Código:

```
package tp3.xmlreadwrite;

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.time.LocalDate;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
  * Esta aplicação serve para exemplificar a utilização de uma API disponibilizada nos packages javax.xml e org.w3.dom para ler
 * um ficheiro XML, editar os dados XML carregados, e voltar a gravar esses dados num ficheiro novo.
 *
 * @version 1.0
  * @author Docentes da Disciplina de Modelação e Programação, LEIM, Instituto Superior de Engenharia de Lisboa
 *
 */
```

```
public class XMLReadWrite {

	public static void main(String[] args) {

		try {

			File inputFile = new File("XML/BaseDados.xml");

			DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
			DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
			Document doc = dBuilder.parse(inputFile);
			System.out.println("Root element :" + doc.getDocumentElement().getNodeName());

			XPath xpath = XPathFactory.newInstance().newXPath();
			String expression = "/catalog/*";
			NodeList nBooks = (NodeList) xpath.evaluate(expression, doc, XPathConstants.NODESET);

			//Mostrar na consola todos os Elementos "book"
			showAllBooks(nBooks);

			//Adicionar um novo "book" à lista nList de "book"s carregados da base de dados (BaseDados.xml)
			Element newBook = createNewBook(doc, "New Book", "003", "Mr Zen", "Fantasy", 5.0f);

			expression = "/catalog";
			NodeList nList = (NodeList) xpath.evaluate(expression, doc, XPathConstants.NODESET);

			Node nCatalog = nList.item(0);

			nCatalog.appendChild(newBook);

			//Escrever, para o documento NewDB, o novo documento actualizado
			FileOutputStream output = new FileOutputStream("XML/NewDB.xml");
		writeXml(doc, output);


		} catch (Exception e) {
			e.printStackTrace();
		}


	}
```

```
	/**
	 * Mostra na consola informações sobre todos os elementos com o nome "book" passados no argumento.
	 * @param nBooks uma lista de nós contendo os elementos com nome "book".
	 */
	public static void showAllBooks(NodeList nBooks) {

		System.out.println("Books in Catalog :");
		for(int index = 0; index < nBooks.getLength(); index++) {

			Node nBook = nBooks.item(index);
			if (nBook.getNodeType() == Node.ELEMENT_NODE) {
				Element eBook = (Element) nBook;
				String id = eBook.getAttribute("id");
				String title = eBook.getElementsByTagName("title").item(0).getTextContent();
				String author = eBook.getElementsByTagName("author").item(0).getTextContent();
				String price = eBook.getElementsByTagName("price").item(0).getTextContent();
				System.out.println("id:" + id + " Title: " + title + ", Autor: " + author + " , price: " + price);
			}
		}

	}
```

```
        /**
         * Cria um novo Elemento "book".
         *
         * @param doc o Documento que irá gerar os novos Elementos
         * @param title o título do "book"
         * @param id  o attributo ID do "book"
         * @param author o autor
         * @param genre o género
         * @param price o preço
           *  @return um novo Element com nome "book" contendo os
dados/subelementos passados como argumento.
         */
        public static Element createNewBook(Document doc, String title, String
id, String author, String genre, float price) {

                Element eBook   = doc.createElement("book");
                eBook.setAttribute("id", id);

                Element eAuthor   = doc.createElement("author");
                eAuthor.appendChild(doc.createTextNode(author));
                eBook.appendChild(eAuthor);

                Element eTitle   = doc.createElement("title");
                eTitle.appendChild(doc.createTextNode(title));
                eBook.appendChild(eTitle);

                Element eGenre   = doc.createElement("genre");
                eGenre.appendChild(doc.createTextNode(genre));
                eBook.appendChild(eGenre);

                Element ePrice = doc.createElement("price");
                ePrice.appendChild(doc.createTextNode(Float.toString(price)));

                eBook.appendChild(ePrice);

                Element eDate = doc.createElement("publish_date");

eDate.appendChild(doc.createTextNode(LocalDate.now().toString()));
                eBook.appendChild(eDate);

                Element eDesc = doc.createElement("description");
                eDesc.appendChild(doc.createTextNode("No Description"));

                eBook.appendChild(eDesc);

                return eBook;
        }
```

```
      /**
       * Escreve, para o OutputStream, o documento doc.
       * @param doc o documento contendo os Elementos a gravar on ficheiro
output
       * @param output o ficheiro de saída.
       */
     private static void writeXml(Document doc, OutputStream output) throws
TransformerException {

                                TransformerFactory    transformerFactory    =
TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();

        // pretty print XML
        transformer.setOutputProperty(OutputKeys.INDENT, "yes");

        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(output);

        transformer.transform(source, result);

    }


}
```

2. tp3XML

2.1. Evento

Para esta classe, foi-nos pedido para implementar apenas três métodos, dado que a classe foi disponibilizada pelos docentes. Duas destas já eram familiares do trabalho prático anterior, o *toString* e o *print*.

A terceira função a ser implementada é a *build* que recebe Node como argumento. O objectivo deste método, nesta classe, é identificar que tipo de Evento é que está escrito no Node e criar esse mesmo, com auxílio aos métodos estáticos das classes Espetaculo e Festival.

Código:

```
package tp3.XML;

import java.util.Arrays;

import org.w3c.dom.*;

/**
 * Classe abstracta que representa um Evento.
 *
 * @version 1.0
  * @author Docentes da Disciplina de Modelação e Programação, LEIM,
Instituto Superior de Engenharia de Lisboa
 *
 */
public abstract class Evento {

	private String nome;

	public Evento(String nome) {this.nome = nome; }

	public String getNome() {return nome; }
	public abstract int getNumBilhetes();
	public abstract String[] getArtistas();
	public abstract int numActuacoes(String artista);

	/**
	 * Cria um novo Elemento contendo todas as informações deste Evento.
	 * @param doc o documento que irá gerar o novo Elemento.
	 * @return um Elemento que represnta o Evento na arvore XML
	 */
	public abstract Element createElement(Document doc);

	/**
	 * Retorna uma String contendo informações sobre o Evento.
	 * Nota: Ver o ficheiro OutputPretendido.txt
	 */
	public String toString() {
		String toString = nome + " com " + getNumBilhetes() + " bilhetes
e com ";

		if(getArtistas().length > 1)
		{
			toString += "os artistas [";
```

```
            }
            else
            {
                  toString += "o artista [";
            }

            for(int index = 0; index < getArtistas().length; index++)
            {
                  toString += getArtistas()[index];

                  if(index != getArtistas().length - 1)
                  {
                        toString += ", ";
                  }
            }

            toString += "]";

            return toString;
      }

      /**
        * Escreve, para a consola, o prefixo seguido da String representativa do Evento.
       * @param prefix - Um prefixo para gerar a identação apropriada de acordo com a "profundidade".
       */
      public void print(String prefix) {
            System.out.println(prefix + toString());
      }

      /**
        * Constroi um novo Evento (Espetáculo ou Festival) a partir das informações existentes no nó
       * nNode que foi lido da arvore XML.
        * @param nNode o nó/elemento contendo a informação do Evento.
       * @return o novo Evento associado a esse nNode.
       */
      public static Evento build(Node nNode) {

            String nome = nNode.getNodeName();
            Evento evento;

            if(nome.equals("Festival"))
            {
                  evento = Festival.build(nNode);
            }
            else
            {
                  evento = Espetaculo.build(nNode);
            }

            return evento;
      }
```

```
}
```

2.2. Espetaculo

Para esta classe, foi-nos pedido para completar todos os métodos existentes. A maior parte destes métodos eram conhecidos do trabalho prático anterior, como o constructor da classe, *numActuacoes, addArtista, getNumBilhetes, getArtistas, getLocalidade* e o *toString*.

Os novos métodos a serem implementados foram o *build*, que recebia um Node e devolvia um novo Espetaculo, baseando-se na informação recebida. E o método *createElemen*t, que cria um elemento de XML com a informação do Espetaculo e este é devolvido.

Código:

```
package tp3.XML;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * Classe que representa um Evento do tipo Espetaculo.
 *
 * @version 1.0
  * @author Docentes da Disciplina de Modelação e Programação, LEIM,
Instituto Superior de Engenharia de Lisboa
 *
 */
public class Espetaculo extends Evento {

	private static final int MAX_ARTISTAS = 10;
	private String[] artistas = new String[MAX_ARTISTAS];
	private int nArtistas = 0;
	private int numBilhetes;
	private String localidade;
```

```
	/**
	 * Constroi um novo Espetaculo
	 * @param nome nome do Espetaculo
	 * @param localidade a localidade do Espetaculo
	 * @param numBilhetes o número de bilhetes disponíveis
	 */
	public Espetaculo(String nome, String localidade, int numBilhetes)
	{
		super(nome);

		//Localidade
		if(localidade == null)
		{
			throw new IllegalArgumentException("A localidade não pode ser null");
		}

		if(localidade.length() == 0)
		{
			throw new IllegalArgumentException("O tamanho da localidade não pode ser 0");
		}

		int letterCounter = 0;

		for(int charIndex = 0; charIndex < localidade.length(); charIndex++)
	{
		if(!Character.isLetter(localidade.charAt(charIndex)) && !Character.isWhitespace(localidade.charAt(charIndex)) && !Character.isDigit(localidade.charAt(charIndex)))
			{
			throw new IllegalArgumentException("A localidade só pode possuir letras, espaços e números");
			}

		if(Character.isLetter(localidade.charAt(charIndex)))
		{
			letterCounter++;
		}
	}

		if(letterCounter == 0)
		{
			throw new IllegalArgumentException("A localidade tem de ter, pelo menos, um caractere");
		}

		this.localidade = localidade;
```

```
            //NumBilhetes
            if(numBilhetes <= 0)
            {
                  throw new IllegalArgumentException("O número de bilhetes
deve ser superior a 0");
            }

            this.numBilhetes = numBilhetes;
      }

      /**
       * Informa se um determinado artista irá actuar no Espetaculo.
       * @return 1, se actuar e 0 caso contrário.
       * @Override
       */
      public int numActuacoes(String artista) {

            for(int artistaIndex = 0; artistaIndex < nArtistas;
artistaIndex++)
            {
                  if(artistas[artistaIndex].equals(artista))
                  {
                        return 1;
                  }
            }

            return 0;
      }

      /**
       * Permite adicionar un novo artista ao Espetaculo se o artista ainda
       * não tem actuações e se o número máximo de artistas ainda não foi
ultrapassado.
       * @param artista representa o novo artista
       * @return verdadeiro, caso o artista tenha sido adicionado e falso
caso contrário.
       */
      public boolean addArtista(String artista) {

            if(nArtistas >= artistas.length || numActuacoes(artista) != 0)
            {
                  return false;
            }

            artistas[nArtistas] = artista;
            nArtistas++;
            return true;
      }
```

```java
/**
 * Devolve o número de bilhetes.
 * @Override
 */
public int getNumBilhetes() {
	return numBilhetes;
}

/**
 * Devolve uma cópia dos artistas que actuam no Espetaculo.
 * @Override
 */
public String[] getArtistas() {
	String[] artistasRegistados = new String[nArtistas];

	for(int index = 0; index < nArtistas; index++)
	{
		artistasRegistados[index] = artistas[index];
	}

	return artistasRegistados;
}

/**
 * Devolve a localidade do Espetaculo
 * @return a localidade.
 */
public String getLocalidade() {
	return localidade;
}

/**
 * Devolve uma String a representar o Espetaculo.
 * Nota: Ver o ficheiro OutputPretendido.txt
* @Override
 */
public String toString() {
	return super.toString() + " em " + localidade;
}
```

```
	/**
	 * Constroi um novo Evento a partir do objecto Node passado como parâmetro.
	 * @param nNode
	 * @return um novo Evento
	 */
	public static Evento build(Node nNode) {

		NodeList composicaoDoNode = nNode.getChildNodes();
		NodeList listaDeArtistas = composicaoDoNode.item(3).getChildNodes();

		String nomeDoEspetaculo = composicaoDoNode.item(1).getTextContent();
		String localidade = composicaoDoNode.item(5).getTextContent();

		int numBilhetes = 0;

		if(nNode.getAttributes().getLength() == 1)
		{
			numBilhetes = Integer.parseInt(nNode.getAttributes().item(0).getTextContent());

		}
		else
		{
			numBilhetes = Integer.parseInt(nNode.getAttributes().item(1).getTextContent());
		}

		Espetaculo espetaculo = new Espetaculo(nomeDoEspetaculo, localidade, numBilhetes);

		for(int artistaIndex = 1; artistaIndex < listaDeArtistas.getLength(); artistaIndex += 2)
		{

			espetaculo.addArtista(listaDeArtistas.item(artistaIndex).getTextContent());
		}

		return espetaculo;
	}
```

```
	/**
	 *  Constroi um novo Element a partir do Espectaculo actual.
	 *  @param doc - o documento que irá gerar o novo Element
	 */
	public Element createElement(Document doc) {

		Element elementoEspetaculo = doc.createElement("Espetaculo");

		elementoEspetaculo.setAttribute("numBilhetes",
Integer.toString(numBilhetes));

		Element elementoNome = doc.createElement("Nome");
		elementoNome.appendChild(doc.createTextNode(getNome()));

		Element elementoArtistas = doc.createElement("Artistas");

		for(int artistaIndex = 0; artistaIndex < nArtistas;
artistaIndex++)
		{
			Element elementoArtista = doc.createElement("Artista");

elementoArtista.appendChild(doc.createTextNode(artistas[artistaIndex]));

			elementoArtistas.appendChild(elementoArtista);
		}

		Element elementoLocalidade = doc.createElement("Localidade");
		elementoLocalidade.appendChild(doc.createTextNode(localidade));

		elementoEspetaculo.appendChild(elementoNome);
		elementoEspetaculo.appendChild(elementoArtistas);
		elementoEspetaculo.appendChild(elementoLocalidade);

		return elementoEspetaculo;
	}
}
```

2.3. Festival

Na classe Festival, foi necessário desenvolver o constructor, *getNumBilhetes, numActuacoes, toString*, *getArtistas, getDeepFestival, addEvento, delEvento* e *print*.

Tal como na classe Espetaculo, foi-nos pedido para desenvolver duas novas funções: *build*, que recebia um Node e devolvia um novo Festival, com a informação presente no Node; e *createElement* que cria um elemento de XML com a informação do Festival e este é devolvido.

Código:

```
package tp3.XML;

import java.util.ArrayList;
import java.util.List;


import org.w3c.dom.*;

import javax.xml.parsers.*;
import javax.xml.xpath.*;
import java.io.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

/**
 * Classe que representa um Evento do tipo Festival.
 *
 * @version 1.0
  * @author Docentes da Disciplina de Modelação e Programação, LEIM,
Instituto Superior de Engenharia de Lisboa
 *
 */
public class Festival extends Evento {

	private static final int MAX_EVENTOS = 20;

	private Evento[] eventos = new Evento[MAX_EVENTOS];
	private int numEventos = 0;

	public Festival(String nome) {
		super(nome);
	}
```

```
	/**
	 * Devolve todos os bilhetes existentes no Festival (somando e
devolvendo todos os bilhetes dos seus Eventos).
	 * @return o número de bilhetes existentes no Festival.
	 */
	public int getNumBilhetes() {

		int numBilhetes = 0;

		for(int index = 0; index < eventos.length; index++)
		{
			if(eventos[index] != null)
			{
				numBilhetes += eventos[index].getNumBilhetes();

			}
		}

		return numBilhetes;
	}

	/**
	 * Retorna o número de actuaçõoes de um determinado artista.
	 * @param o nome do artista.
	 * @Override
	 */
	public int numActuacoes(String artista) {

		int numActuacoes = 0;

		for(int index = 0; index < eventos.length; index++)
		{
			if(eventos[index] != null)
			{
				numActuacoes                                    +=
eventos[index].numActuacoes(artista);
			}
		}

		return numActuacoes;
	}

	/**
	 *  Devolve uma string representativa do Festival.
	 *  Nota: Ver o ficheiro OutputPretendido/OutputPretendido.txt
	 */
	public String toString() {
		return "Festival " + super.toString();
	}
```

```
	/**
	 * Devolve um array contendo todos, de forma não repetida, os nomes
de todos os artistas quer irão
	 * actuar no Festival.
	 * @return um array contendo os nomes dos artistas.
	 */
	public String[] getArtistas() {

		List<String> artistas = new ArrayList<String>();

		for(int index = 0; index < eventos.length; index++)
		{
			if(eventos[index] != null)
			{
				String[] artistasRecolhidos =
eventos[index].getArtistas();

				for(int indexArtistas = 0; indexArtistas <
artistasRecolhidos.length; indexArtistas++)
				{

if(!artistas.contains(artistasRecolhidos[indexArtistas]))
					{

artistas.add(artistasRecolhidos[indexArtistas]);
					}
				}
			}
		}

		return (String[]) artistas.toArray(new String[artistas.size()]);
	}
```

```
	/**
	 * Retorna a profundidade maxima da "árvore" que contém Festivais dentro de Festivais. O próprio Festival não conta.
	 * @return a profundidade máxima.
	 */
	public int getDeepFestival() {

		int profundidade = -1;

		for(int index = 0; index < eventos.length; index++)
		{
			if(eventos[index] instanceof Festival)
			{
				profundidade = Math.max(profundidade, ((Festival) eventos[index]).getDeepFestival());
			}
		}

		profundidade++;

		return profundidade;
	}
```

```
	/**
	 * Adiciona um novo Evento ao Festival, caso para nenhum dos artistas do novo evento se verifique que o seu número de atuações no
	 * novo evento (a adicionar) supere em mais de duas o número de atuações no festival corrente.
	 * @param evento
	 * @return verdadeiro, se o novo Evento foi adicionado.
	 */
	public boolean addEvento(Evento evento) {

		if(numEventos == eventos.length || evento == null)
		{
			return false;
		}

		if(evento.getArtistas().length == 0)
		{
			return false;
		}

		String[] artistasEvento = evento.getArtistas();

		for(int indexArtista = 0; indexArtista < getArtistas().length; indexArtista++)
		{
			String artista = getArtistas()[indexArtista];

			for(int indexAComparar = 0; indexAComparar < artistasEvento.length; indexAComparar++)
			{
				String artistaAComparar = artistasEvento[indexAComparar];

				if(artista.equals(artistaAComparar))
				{
					if(evento.numActuacoes(artistaAComparar) > numActuacoes(artista) + 2)
					{
						return false;
					}
				}
			}
		}
```

```
        for(int index = 0; index < eventos.length; index++)
        {
              if(eventos[index] == null)
              {
                    eventos[index] = evento;
                    numEventos += 1;
                    break;
              }
        }

        return true;

}
```

```
	/**
	 * Remove um evento em qualquer profundidade do Festival corrente.
	 * @param nomeEvento nome do Evento a remover.
	 * @return verdadeiro, se o Evento foi removido.
	 */
	public boolean delEvento(String nomeEvento) {

		if(numEventos == 0 || nomeEvento == null)
		{
			return false;
		}

		boolean removido = false;

		for(int index = 0; index < eventos.length; index++)
		{
			if(eventos[index] instanceof Espetaculo)
			{
				if(eventos[index].toString().contains(nomeEvento))
				{
					eventos[index] = null;
					numEventos--;
					removido = true;
				}
			}
			else if(eventos[index] instanceof Festival)
			{
				if(eventos[index].toString().contains(nomeEvento))
				{
					eventos[index] = null;
					numEventos--;
					removido = true;
				}
				else
				{
					boolean removidoRecursivo = ((Festival)
eventos[index]).delEvento(nomeEvento);

					if(!removido && removidoRecursivo)
					{
						removido = removidoRecursivo;
					}
				}
			}
		}

		return removido;
	}
```

```
	/**
	 * Imprime na consola informações sobre o Festival.
		* Nota: Ver o output pretendido em OutputPretendido/OutputPretendido.txt.
	 * @param o prefixo para identar o Festival de acordo com a sua profundidade.
	 */
	public void print(String prefix) {

		System.out.println(prefix + toString());

		for(int eventoIndex = 0; eventoIndex < eventos.length; eventoIndex++)
		{
			if(eventos[eventoIndex] != null)
			{
				eventos[eventoIndex].print(prefix + " ");
			}
		}
	}

	/**
	 * Constroi um novo Festival a partir de um nó contendo as infomações lidas do documento XML.
	 * @param nNode o nó associado ao Festival
	 * @return um novo Festival
	 */
	public static Festival build(Node nNode) {

		NodeList composicaoDoNode = nNode.getChildNodes();

		String nomeFestival = composicaoDoNode.item(1).getTextContent();

		Festival festival = new Festival(nomeFestival);

		if(composicaoDoNode.item(3).hasChildNodes())
		{
			NodeList eventos = composicaoDoNode.item(3).getChildNodes();

			for(int eventosIndex = 1; eventosIndex < eventos.getLength(); eventosIndex += 2)
			{

festival.addEvento(Evento.build(eventos.item(eventosIndex)));
			}
		}

		return festival;
	}
```

```
	/**
	 * Cria um novo Elemento quer irá representar, no documento XML, o Festival associado ao Festival corrente.
	 * @param doc o Documento que irá ser usado para gerar o novo Element.
	 */
	public Element createElement(Document doc) {

		Element elementoFestival = doc.createElement("Festival");

		Element elementoNome = doc.createElement("Nome");
		elementoNome.appendChild(doc.createTextNode(getNome()));

		Element elementoEventos = doc.createElement("Eventos");

		for(int indexEventos = 0; indexEventos < eventos.length; indexEventos++)
		{
			if(eventos[indexEventos] != null)
			{
				Element elementoEvento = eventos[indexEventos].createElement(doc);

				elementoEventos.appendChild(elementoEvento);
			}
		}

		elementoFestival.appendChild(elementoNome);
		elementoFestival.appendChild(elementoEventos);

		return elementoFestival;
	}
```

```
	/**
	 * Método main que gera no output o que está no ficheiro OutputPretendido/OutputPretendido.txt e cria um novo
	 * documento XML/Eventos.xml, com a mesma estrutura que o documento OutputPretendido/Eventos.xml.
	 * @param args
	 */
	public static void main(String[] args) {

		try {

			File inputFile = new File("XML/BaseDados.xml");

			DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
			DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
			Document doc = dBuilder.parse(inputFile);
			System.out.println("Root element :" + doc.getDocumentElement().getNodeName());

			XPath xpath = XPathFactory.newInstance().newXPath();
			String expression = "/BaseDados/Eventos/*";
			NodeList nList = (NodeList) xpath.evaluate(expression, doc, XPathConstants.NODESET);

			Node nNode = nList.item(0);
			Evento evento = Evento.build(nNode);
			if(evento != null) evento.print("");


			Festival fNovo = new Festival("Bollywood Music Festival");

			Espetaculo e1_1 = new Espetaculo("Suna Hai", "Sines", 500);
			e1_1.addArtista("Suna Hai");
			fNovo.addEvento(e1_1);

			Espetaculo e1_2 = new Espetaculo("Rait Zara", "Sines", 400);
			e1_2.addArtista("Rait Zara");
			fNovo.addEvento(e1_2);

			if(evento instanceof Festival) {

				Festival festival = (Festival)evento;
				festival.addEvento(fNovo);
```

```
                        // root elements
                        Document newDoc = dBuilder.newDocument();
                                Element rootElement =
newDoc.createElement("Eventos");


rootElement.appendChild(festival.createElement(newDoc));

                        newDoc.appendChild(rootElement);

                                FileOutputStream output = new
FileOutputStream("XML/Eventos.xml");
                    writeXml(newDoc, output);

                    }

            } catch (Exception e) {
               e.printStackTrace();
            }

      }

      /**
       * Escreve, para o OutputStream, o documento doc.
       * @param doc o documento contendo os Elementos a gravar on ficheiro
output
       * @param output o ficheiro de saída.
       */
      private static void writeXml(Document doc, OutputStream output) throws
TransformerException {

                                TransformerFactory transformerFactory =
TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();

        // pretty print XML
        transformer.setOutputProperty(OutputKeys.INDENT, "yes");

        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(output);

        transformer.transform(source, result);
    }


}
```

3. tp3XPATH

3.1. Solucoes

Nesta parte do trabalho, é-nos pedido para arranjarmos alguns elementos do documento XML *BaseDados.xml*.

a) //Espetaculo[@numBilhetes < 500]

b) //Festival/Nome[text()='Músicas do Mundo']/../Eventos/*[1]/Nome/text()

c) //Artista[text()="Pablo Milanés"]/../..

d) count(//Festival/Nome[text()="Música Cubana"]/../Eventos/*)

e) //*[@codEspetaculo=//Patrocinador/Nome[text()="EDP"]/../@codEvento]

f) //Festival/Nome[text()="Músicas do Mundo"]/../Eventos//Festival/
Nome[text()="Música Cubana"]/../Eventos//
Nome[text()="Pablo Milanés"]/../following-sibling::Espetaculo